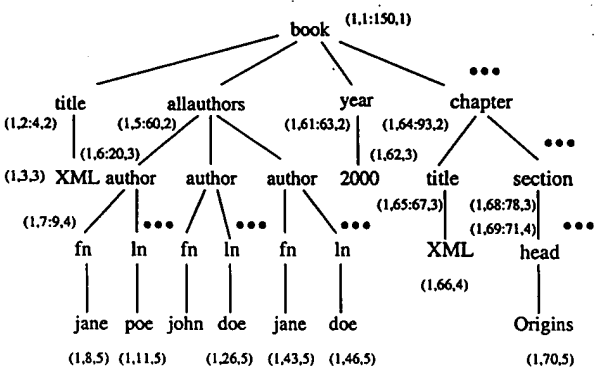

book
(1,1:150,1)
title
(1,2:4,2)
allauthors
(1,5:60,2)
year
(1,61:63,2)
chapter
(1,64:93,2)
(1,6:20,3)
(1,62,3)
(1,3,3)
XML
author
author
author
2000
title
section
(1,65:67,3)
(1,68:78,3)
(1,69:71,4)
(1,7:9,4)
fn
ln
fn
ln
fn
ln
XML
head
(1,66,4)
jane
poe
john
doe
jane
doe
Origins
(1,8,5)
(1,11,5)
(1,26,5)
(1,43,5)
(1,46,5)
(1,70,5)

FIG. 1

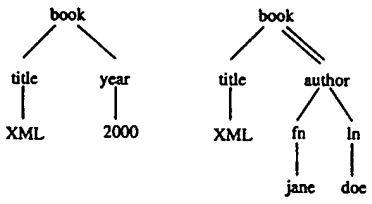


FIG. 2A FIG. 2B

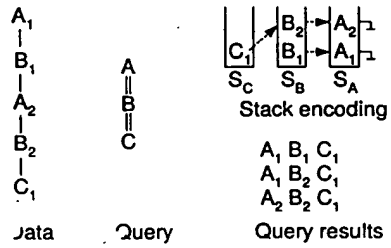


FIG. 3C

FIG. 3A FIG. 3B FIG. 3D

```
Algorithm PathStack(q)
01 while ¬end(q)
02   q_min = getMinSource(q)
03   for q_i in subtreeNodes(q) // clean stacks
04     while (¬empty(S_qi) ∧ topR(S_qi) < nextL(T_qmin))
05       pop(S_qi)
06   moveStreamToStack(T_qmin, S_qmin, pointer to
                                    top(S_parent(qmin)))
07   if (isLeaf(q_min))
08     showSolutions(S_qmin, 1)
09     pop(S_qmin)

Function end(q)
   return ∀q_i ∈ subtreeNodes(q) : isLeaf(q_i) ⇒ eof(T_qi)

Function getMinSource(q)
   return q_i ∈ subtreeNodes(q) such that nextL(T_qi)
     is minimal

Procedure moveStreamToStack(T_q, S_q, p)
01 push(S_q,(next(T_q),p))
02 advance(T_q)
```

PathStack

FIG. 4

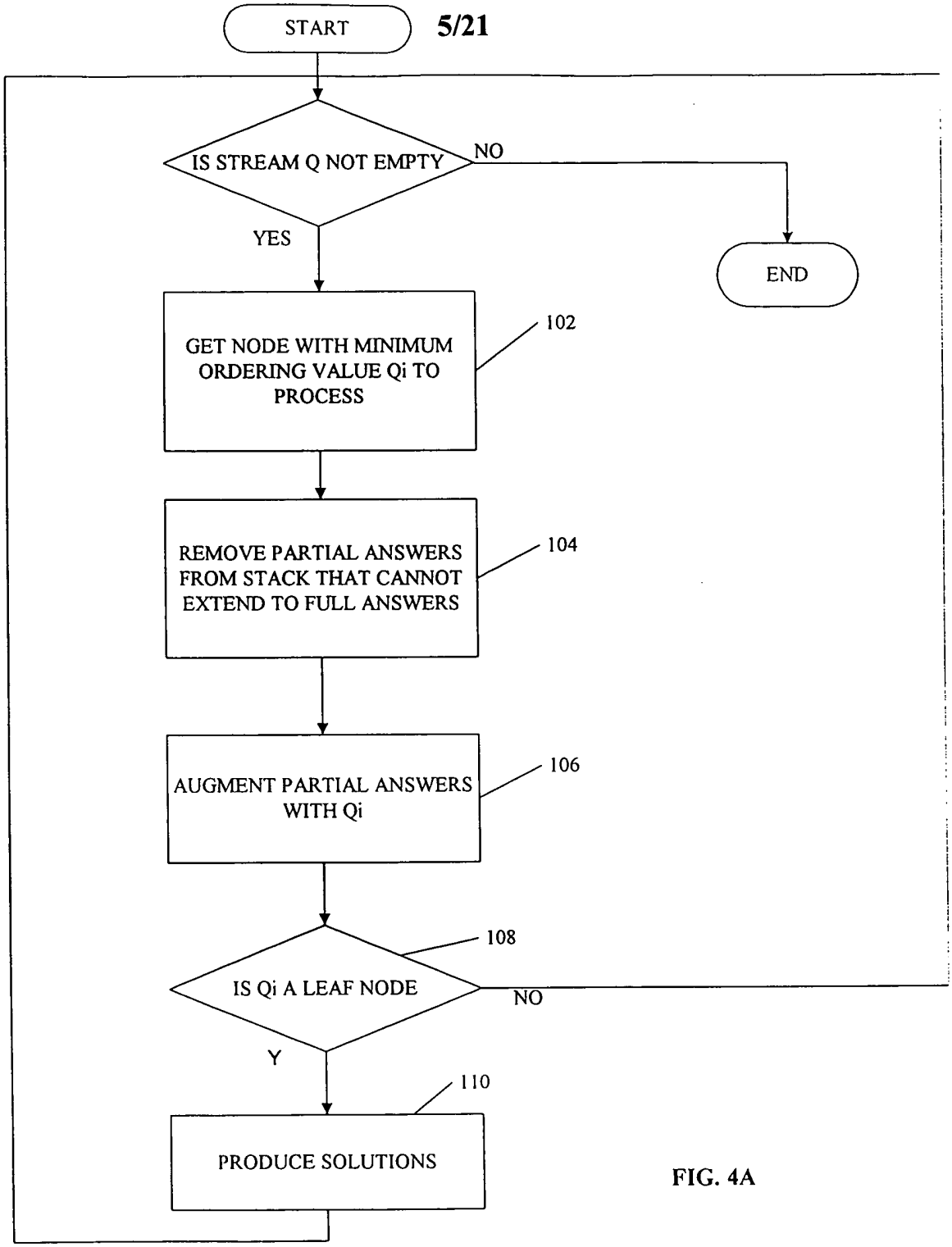

START
IS STREAM Q NOT EMPTY
NO
YES
END
GET NODE WITH MINIMUM ORDERING VALUE Qi TO PROCESS
102
REMOVE PARTIAL ANSWERS FROM STACK THAT CANNOT EXTEND TO FULL ANSWERS
104
AUGMENT PARTIAL ANSWERS WITH Qi
106
IS Qi A LEAF NODE
108
NO
Y
PRODUCE SOLUTIONS
110

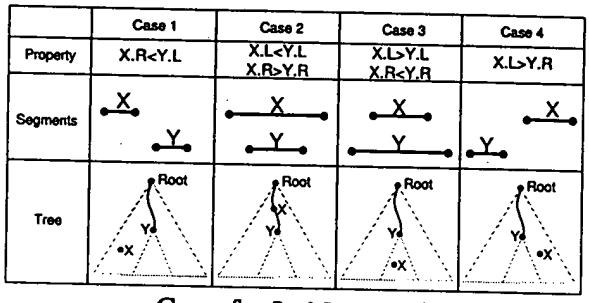
FIG. 4A

```
Procedure showSolutions(SN, SP)
// Assume, for simplicity, that the stacks of the query
//   nodes from the root to the current leaf node we
//   are interested in can be accessed as S[1], ..., S[n].
// Also assume that we have a global array index[1..n]
//   of pointers to the stack elements.
// index[i] represents the position in the i'th stack that
//   we are interested in for the current solution, where
//   the bottom of each stack has position 1.

// Mark we are interested in position SP of stack SN.
01 index[SN] = SP
02 if (SN == 1) // we are in the root
03   // output solutions from the stacks
04   output (S[n].index[n], ..., S[1].index[1])
05 else // recursive call
06   for i = 1 to S[SN].index[SN].pointer_to_parent
07     showSolutions(SN - 1, i)
```

Procedure showSolutions

FIG 5



Cases for PathStack and TwigStack

FIG. 6

```
Algorithm PathMPMJ(q)
01 while (¬eof(T_q)∧ (isRoot(q)∨
                        nextL(q) < nextR(parent(q))))
02   for (q_i ∈ subtreeNodes(q)) // advance descendants
03     while (nextL(q_i) < nextL(parent(q_i)))
04       advance(T_{q_i})
05     PushMark(T_{q_i})
06   if (isLeaf(q)) // solution in the streams' heads
       outputSolution()
07   else PathMPMJ(child(q))
08   advance(T_q)
09   for (q_i ∈ subtreeNodes(q)) // backtrack descendants
10     PopMark(T_{q_i})
```

PathMPMJ

FIG. 7

```
Algorithm TwigStack(q)
  // Phase 1
01 while ¬end(q)
02   q_act = getNext(q)
03   if (¬isRoot(q_act))
04     cleanStack(parent(q_act), nextL(q_act))
05   if (isRoot(q_act) ∨ ¬empty(S_parent(q_act)))
06     cleanStack(q_act, next(q_act))
07     moveStreamToStack(T_q_act, S_q_act, pointer to
                                    top(S_parent(q_act)))
08     if (isLeaf(q_act))
09       showSolutionsWithBlocking(S_q_act, 1)
10       pop(S_q_act)
11   else advance(T_q_act)
  // Phase 2
12 mergeAllPathSolutions()

Function getNext(q)
01 if (isLeaf(q)) return q
02 for q_i in children(q)
03   n_i = getNext(q_i)
04   if (n_i ≠ q_i) return n_i
05 n_min = minarg_n_i  nextL(T_n_i)
06 n_max = maxarg_n_i  nextL(T_n_i)
07 while (nextR(T_q) < nextL(T_n_max))
08   advance(T_q)
09 if (nextL(T_q) < nextL(T_n_min)) return q
10 else return n_min

Procedure cleanStack(S, actL)
01 while (¬empty(S) ∧ (topR(S) < actL))
02   pop(S)
```

TwigStack

FIG. 8

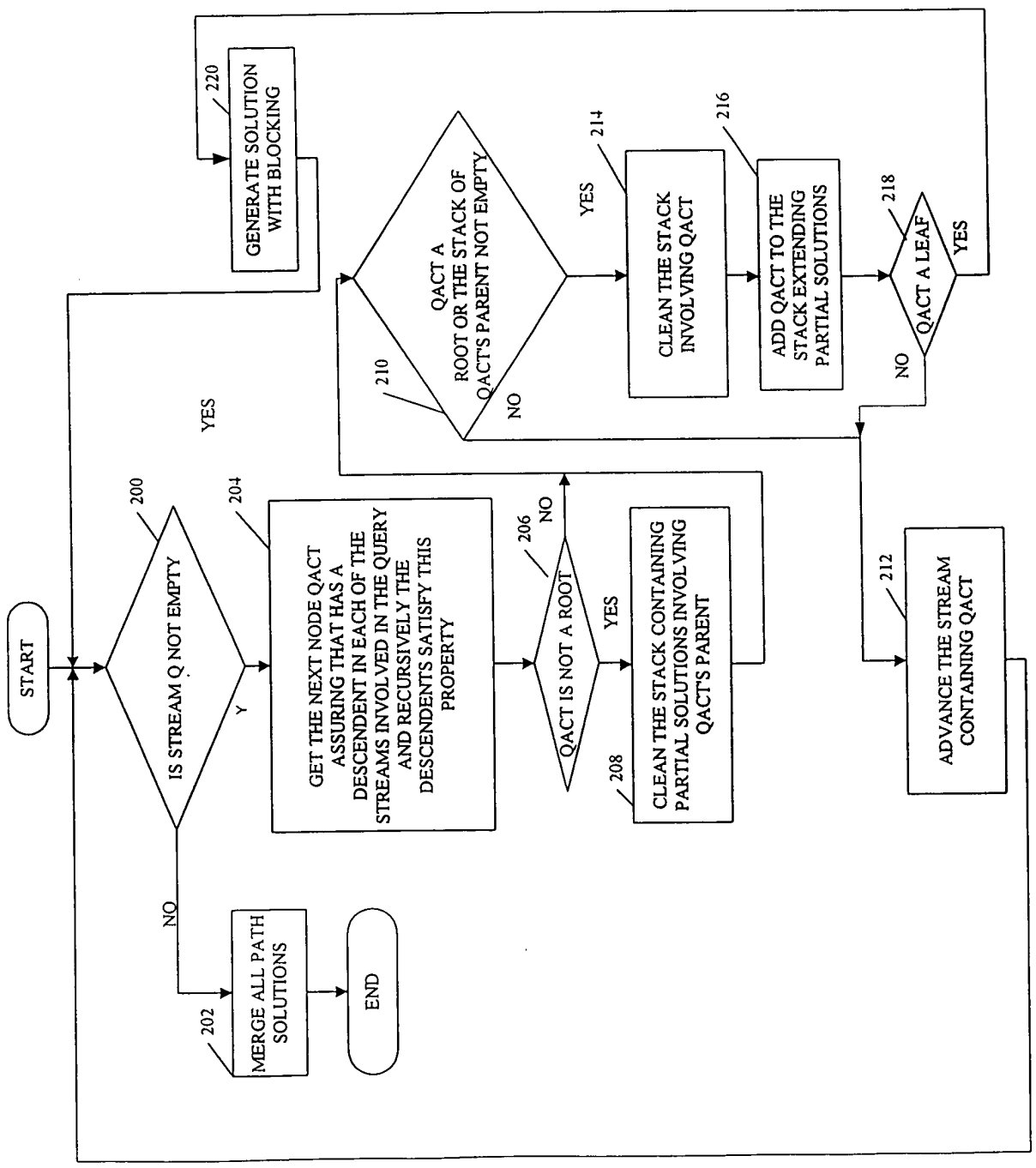

START
200
IS STREAM Q NOT EMPTY
Y
NO
YES
202
MERGE ALL PATH SOLUTIONS
END
204
GET THE NEXT NODE QACT ASSURING THAT HAS A DESCENDENT IN EACH OF THE STREAMS INVOLVED IN THE QUERY AND RECURSIVELY THE DESCENDENTS SATISFY THIS PROPERTY
206
QACT IS NOT A ROOT
NO
YES
208
CLEAN THE STACK CONTAINING PARTIAL SOLUTIONS INVOLVING QACT'S PARENT
210
QACT A ROOT OR THE STACK OF QACT'S PARENT NOT EMPTY
NO
YES
214
CLEAN THE STACK INVOLVING QACT
216
ADD QACT TO THE STACK EXTENDING PARTIAL SOLUTIONS
218
QACT A LEAF
NO
YES
212
ADVANCE THE STREAM CONTAINING QACT
220
GENERATE SOLUTION WITH BLOCKING

FIG. 8A

```
Algorithm TwigStackXB(q)
01 while ¬end(q)
02   q_act = getNext(q)
(03) if (isPlainValue(T_{q_act}))
04     if (¬isRoot(q_act))
05       cleanStack(parent(q_act), next(q_act))
06     if (isRoot(q_act) ∨ ¬empty(S_{parent(q_act)}))
07       cleanStack(q_act, next(q_act))
08       moveStreamToStack(T_{q_act}, S_{q_act}, pointer to
                                        top(S_{parent(q_act)}))
09       if (isLeaf(q_act))
10         showSolutionsWithBlocking(S_{q_act}, 1)
11         pop(S_{q_act})
12     else advance(T_{q_act})
(13) else if (¬isRoot(q_act) ∧ empty(S_{parent(q_act)})∧
              nextL(T_{parent(q_act)}) > nextR(T_{q_act}))
(14)   advance(T_{q_act}) // Not part of a solution
(15) else // Might have a child in some solution
(16)   drillDown(T_{q_act})
   // Phase 2
17 mergeAllPathSolutions()

Function getNext(q)
01 if (isLeaf(q)) return q
02 for q_i in children(q)
03   n_i = getNext(q_i)
(04) if (q_i ≠ n_i ∨ ¬isPlainValue(T_{n_i})) return n_i
05 n_min = minarg_{n_i} nextL(T_{n_i})
06 n_max = maxarg_{n_i} nextL(T_{n_i})
07 while (nextR(T_q) < nextL(T_{n_max}))
08   advance(T_q)
09 if (nextL(T_q) < nextL(T_{n_min})) return q
10 else return n_min

Procedure cleanStack(S, actL)
01 while (¬empty(S) ∧ (topR(S) < actL))
02   pop(S)
```

TwigStackXB

FIG. 9

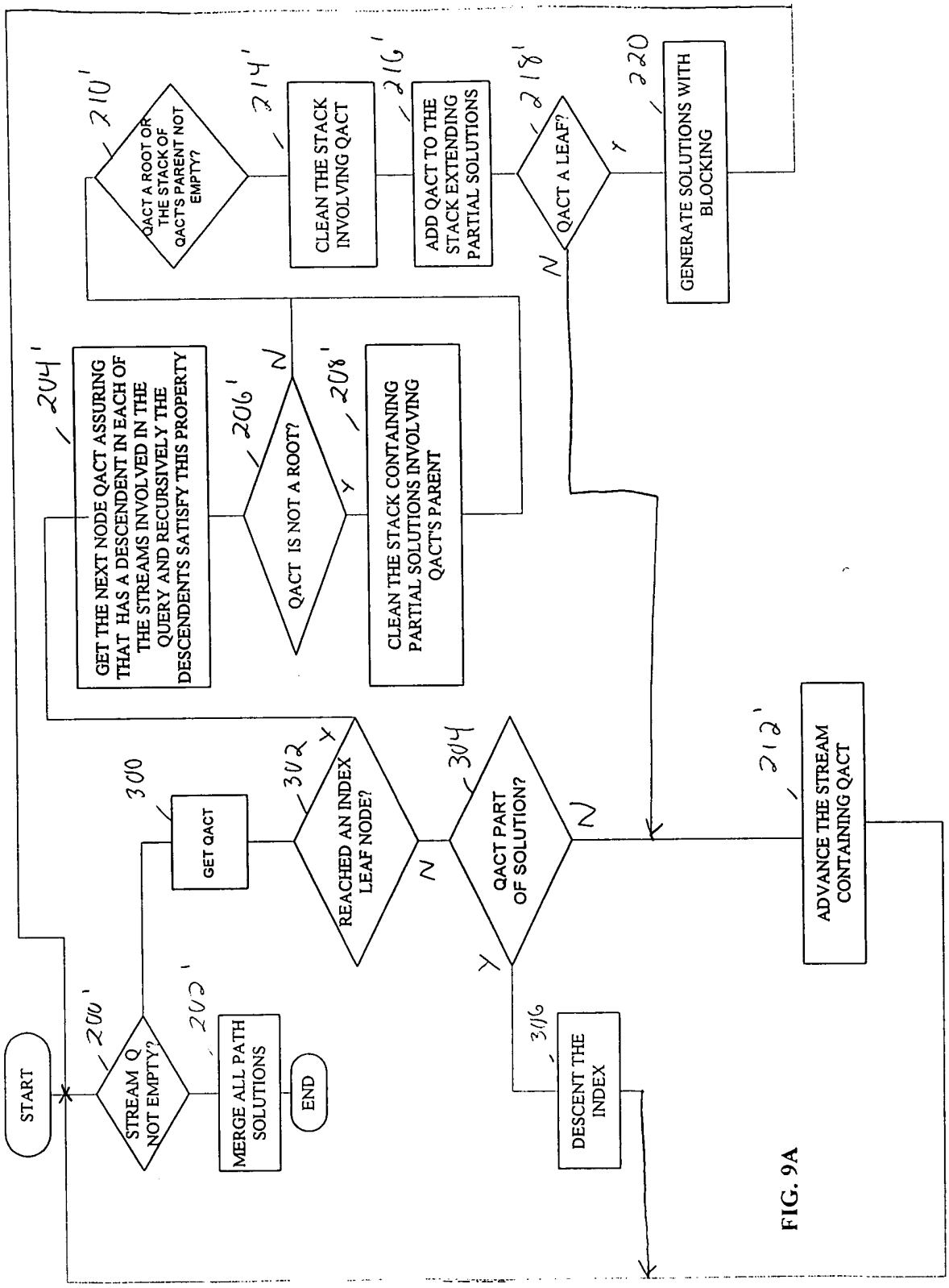
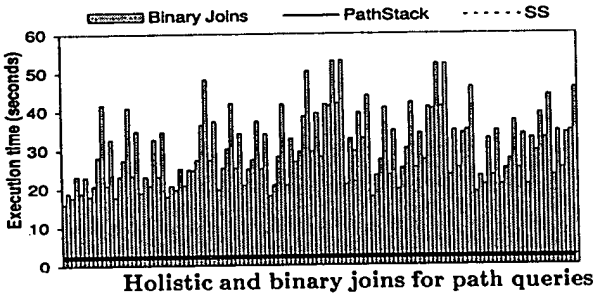
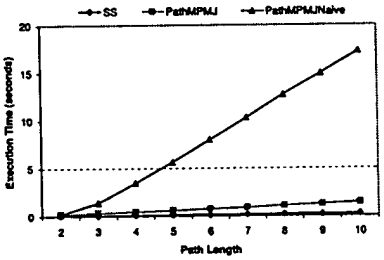

START
200'
STREAM Q NOT EMPTY?
202'
MERGE ALL PATH SOLUTIONS
END
300
GET QACT
302
REACHED AN INDEX LEAF NODE?
304
QACT PART OF SOLUTION?
306
DESCENT THE INDEX
212'
ADVANCE THE STREAM CONTAINING QACT
204'
GET THE NEXT NODE QACT ASSURING THAT HAS A DESCENDENT IN EACH OF THE STREAMS INVOLVED IN THE QUERY AND RECURSIVELY THE DESCENDENTS SATISFY THIS PROPERTY
206'
QACT IS NOT A ROOT?
208'
CLEAN THE STACK CONTAINING PARTIAL SOLUTIONS INVOLVING QACT'S PARENT
210'
QACT A ROOT OR THE STACK OF QACT'S PARENT NOT EMPTY?
214'
CLEAN THE STACK INVOLVING QACT
216'
ADD QACT TO THE STACK EXTENDING PARTIAL SOLUTIONS
218'
QACT A LEAF?
220
GENERATE SOLUTIONS WITH BLOCKING
Y
N

FIG. 9A



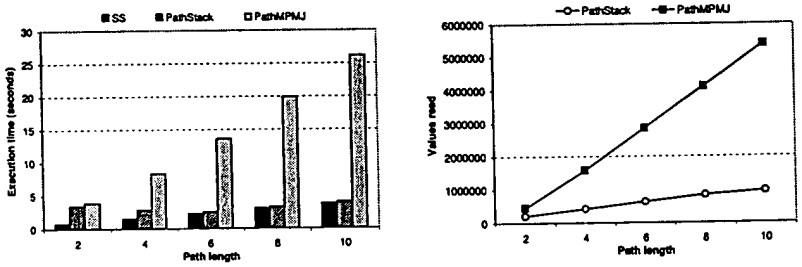
Holistic and binary joins for path queries

FIG. 10



PathMPMJ versus PathMPMJNaive

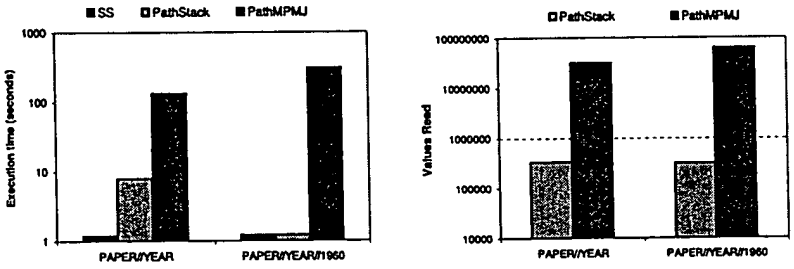
FIG. 11



(a) Execution time (b) Number of elements read

PathStack versus PathMPMJ using synthetic data sets

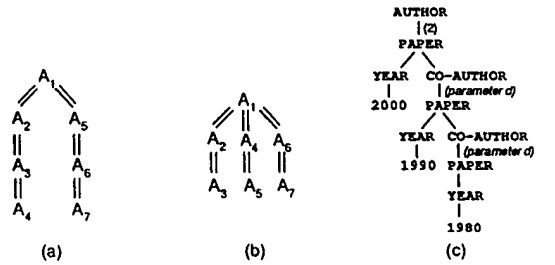
FIG. 12A FIG. 12B



(a) Execution time (b) Number of elements read

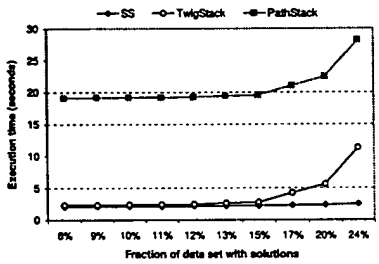
PathStack versus PathMPMJ for the unfolded DBLP data set

FIG. 13A FIG 13B

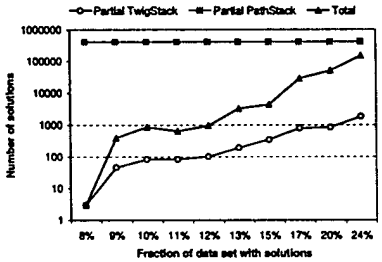


Twig queries used in the experiments

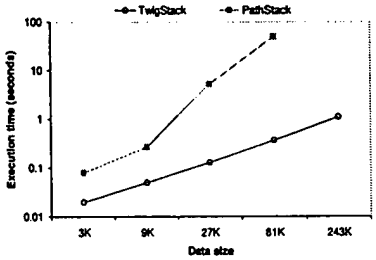
FIG. 14A FIG. 14B FIG. 14C



(a) Execution time



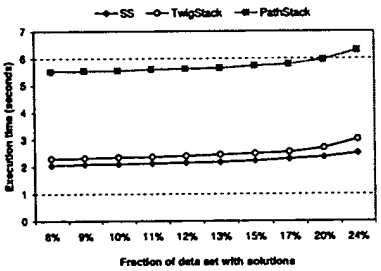
(b) Number of solutions



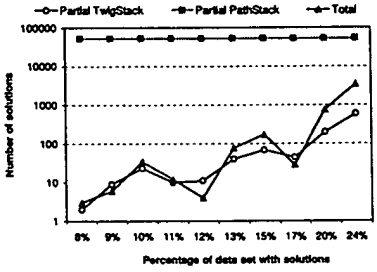
(c) Execution time for a complex query

PathStack versus TwigStack for two twig queries

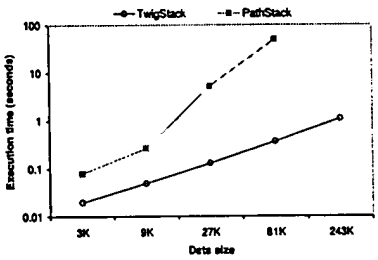
FIG 15A FIG 15B FIG 15C



(a) Execution time



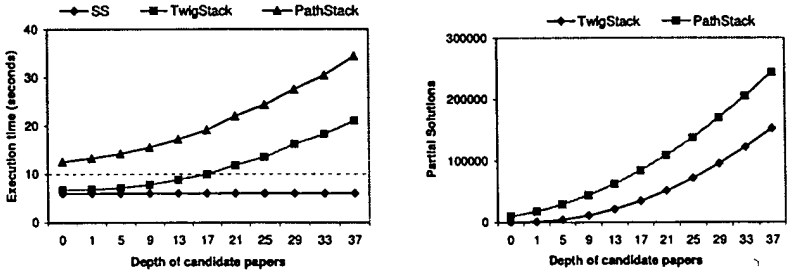
(b) Number of solutions



(c) Execution time for a complex query

PathStack versus TwigStack for a parent-child twig query

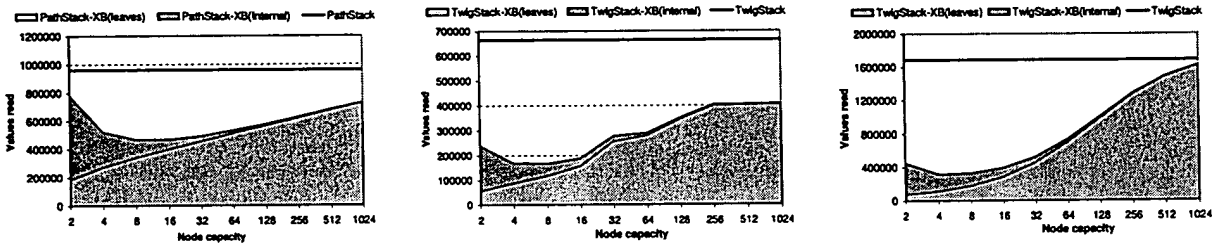
FIG. 16A FIG 16B FIG 16C



(a) Execution time (b) Number of partial solutions

PathStack versus TwigStack on a real data set

FIG. 17A FIG 17B



(a) Path query (b) Twig query (c) Twig query

Using XB-trees: (a), (b) with synthetic data sets, and (c) with unfolded DBLP data

FIG. 18A FIG 18B FIG. 18C